Fast and scalable virtual machine deployment

Razavi, K.

2015

document version
Publisher's PDF, also known as Version of record

Summary

Compared to a decade ago, in a data center today, virtual machines (VMs) have replaced UNIX processes due to their stronger isolation and additional flexibility for their users. Similarly, VM images have replaced program binaries. A VM image is a block-level representation of a VM's root drive, containing all the dependencies of an application that runs on the cloud, down to the operating system (OS) kernel. VM images are often large (multiple GBs), and unlike program binaries, it is not possible to store all of them on the compute hosts of a data center. Hence, while starting a new VM on a host, the contents of the VM image should be delivered to the host over the network. Further, starting a new VM requires the booting of its OS first that is much slower than starting a new UNIX process (i.e., fork).

Delivery of the VM image contents and the slow booting of the guest OS result in slow VM deployments, often minutes if not more. Even worse, starting many VMs can create scalability bottlenecks, adding significantly to the time it takes for the newly created VMs to become ready. Apart from hampering the usability of cloud infrastructures, slow VM deployments are specifically problematic for elastic cloud applications that require quick access to new VMs, such as autoscaling systems or load balancers, and interactive cloud workloads that deploy VMs in response to a user's request, such as iterative high-performance computing. In this dissertation, we tackled the problem of slow VM deployments and showed how to achieve fast and scalable VM startup. Our solution is based on caching of the VM image contents, deduplication combined with compression for efficient caching, and removing the slow booting of the guest OS from the VM deployment path. We briefly describe them below:

Eager transfer of VM images (i.e., copying VM images from storage servers to compute hosts) requires massive amounts of network transfers, and hence, it is neither scalable nor fast. We showed that on-demand lazy transfers (i.e., dynamically

reading parts of the VM images) during VM deployment do not scale as well, but require less network bandwidth compared to eager transfers. We solved this problem, also known as the "boot storm", for lazy transfers by means of host-side block-level caching. We introduced the notion of "OS boot working set", and showed that these sets are orders of magnitude smaller than the VM images. Due to their small size, these sets make good candidates for caching, hence our chosen name: VM image caches. By storing these caches on a storage medium closer to the VMs (e.g., at the hosts), we can significantly reduce the load on both the network and the storage solutions used for delivering VM images to achieve scalable VM deployment. Our solution shows that our caching technique reduces VM deployment times by removing the per-transfer network delay of lazy VM deployments. As a result, we reduce the deployment time of any number of VMs to that of a single VM that is booting from local storage.

While our VM image caches resolve the scalability issues of VM deployments, their storage at the compute hosts is not scalable, specially in a public cloud with thousands of VM images. To address this issue, we designed Squirrel, a storage system designed to keep large quantities of the VM image caches at the hosts. We studied the effect of deduplication (i.e., a common technique to remove duplicate data), combined with compression (e.g., gzip) on 607 VM images and VM image caches of Windows Azure[1], a public data center, and showed that Squirrel can store a significant number of caches within a very moderate amount of resources at the hosts, without affecting the starting time of VMs. Squirrel showed that it is possible to keep warm caches available all the time without any elaborate cache or VM placement strategy.

For the efficient storage of VM image caches, Squirrel relies on the deduplication feature of the ZFS file system. ZFS and similar deduplication systems perform *global* deduplication by keeping the entire digests from the contents of VM images in main memory for finding every possible duplication. To reduce the main memory requirements of global deduplication, we introduced *local* deduplication. The local deduplication scheme finds duplications in a subset of VM images at each time. We discussed unique properties of VM images that make them good candidates for local deduplication. Using the 1011 VM images and VM image caches from the Windows Azure repository[2], Nuts, our prototype implementation of local deduplication, can achieve more than 80% efficiency of global deduplication, while reducing the main memory requirements by up to 35.3x. The low-overhead nature of Nuts makes it a perfect deduplication engine for caching architectures such as Squirrel, or in general for efficient storage of VM images without requiring access to large amounts of main memory.

With a caching system such as Squirrel that eradicates network and storage bottlenecks, the bulk of the VM startup time is spent booting the OS. To move the slow booting of the guest OS from the fast VM deployment path, we introduced μVM,

[1]This repository includes the public VM images registered by the users from April to November 2013.
[2]This repository includes the public VM images registered by the users from April 2013 to June 2014.

a snapshot of an entire OS with minimal resources (e.g., core, memory, etc.) at the time when it has finished booting. Our VM Bakery service can then extend a resumed µVM to a size requested by the user via hot-plugging resources. By modifying Squirrel to serve µVMs instead of VM image caches, we show that we can deploy VMs in under one second on average using a standard ext4 file system with a cold page-cache, and under 200 milliseconds with a warm page-cache. Caching 1011 µVMs from the public VM images of Windows Azure on a deduplicated and compressed file system amounts to 50 GB of disk space, while maintaining deployment times of 2.8 seconds on average.

The solutions presented in this dissertation provide cloud infrastructures with a similar functionality to a UNIX utility such as `mpirun` that can instantly start processes on many hosts. We expect our results to improve the experience of cloud users and developers, and add to the reliability of cloud applications that require quick access to newly deployed VMs.